

Nº Orden	Apellido y nombre	L.U.	Cantidad de hojas
			4

## Organización del Computador 2

### Primer parcial – 05-05-2015

1 (40)	2 (40)	3 (20)	
--------	--------	--------	--

#### Normas generales

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Está permitido tener los manuales y los apuntes con las listas de instrucciones en el examen. Está prohibido compartir manuales o apuntes entre alumnos durante el examen.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido y LU.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

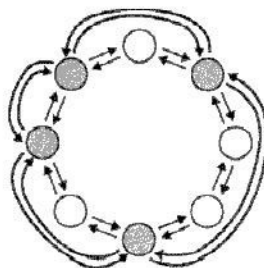
### Ej. 1. (40 puntos)

Se tiene una estructura de lista circular que almacena información sobre procesos. La misma es una lista doblemente encadenada entre un conjunto de nodos, que a su vez contienen otra lista doblemente encadenada entre algunos de estos nodos. Esta última también es circular.

```
struct node {
    int id;
    node* next;
    node* prev;
    node* superNext;
    node* superPrev;
    info* data;
}
```

La estructura contiene dos pares de punteros: **next** y **prev** que construyen la lista circular entre todos los nodos, y **superNext** y **superPrev** que construyen la otra lista entre algunos de estos nodos denominada *super*.

Además, los nodos contienen un puntero **data** que es utilizado para almacenar la información sobre el proceso, y un identificador denominado **id**.

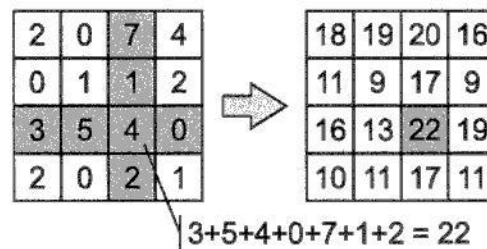


- (10p) (a) Implementar en ASM la función `borrar (info* borrar(node* n))`, que borra el nodo indicado manteniendo el invariante de las dos listas y retorna el puntero a la información que el nodo contenía.
- (15p) (b) Implementar en ASM la función `agregarSuper (void agregarSuper(node* n))`, que toma un nodo *n* que pertenece a la lista y lo agrega a la lista *super*.
- (15p) (c) Implementar en ASM la función `estanTodos (int estanTodos(node* n))`, que retorna 1 si la lista *super* contiene a todos los nodos.

Nota: La lista circular tiene al menos 31 nodos y la lista *super* contiene al menos 21 nodos antes de llamar a cualquiera de las funciones.

## Ej. 2. (40 puntos)

Se tiene una matriz de  $4 \times 4$  enteros sin signo de 32bits. Se desea implementar una función denominada **sumatorias** que se encarga de generar una nueva matriz que contiene por cada elemento la sumatoria de todos los elementos en la fila y columna de la matriz original, contando solo una vez cada uno de los números.



- (30p) (a) Implementar en ASM utilizando instrucciones de SIMD la función **sumatorias**. Procesar la mayor cantidad posible de elementos, justificar.
- (10p) (b) Considerando que la matriz de entrada no contiene enteros, sino *floats* y que en la nueva matriz resultado se deben almacenar enteros de 32bits, modifique el código anterior para reflejar este cambio (puede reescribir todo el código o una parte del mismo).

## Ej. 3. (20 puntos)

Por cuestiones de seguridad se busca implementar una nueva convención para la utilización de la pila. En esta nueva convención se poseen dos pilas independientes que almacenan por un lado direcciones de retorno y por otro, datos.

- (10p) (a) Explique como implementaría una convención con estas características. ¿Utilizaría variables globales?. Explique detalladamente como construir y deshacer el *stack-frame* y cómo llamar a una función. Utilice gráficos y código según corresponda. Recordar que se debe respetar la convención C y la alineación de memoria.
- (10p) (b) Implementar en ASM la siguiente función C, respetando la convención explicada en el punto anterior.

```
convertAndAdd(int a, int b) {
    x = convert(a);
    y = convert(b);
    z = x + y;
    return z;
}
```

1/4

1) ~~node~~

~~node~~ ; Offsets

%define OFF\_N 8

%define OFF\_P 16

%define OFF\_SN 24

%define OFF\_SP 32

%define OFF\_D 40

~~node~~

2) ~~node~~ ; rdi = \*node

mov rax, [rdi + OFF\_SP]

cmp rax, 0

JZ .endSuper ; Cambiar superps

mov rsi, [rdi + OFF\_SN]

mov ~~rax~~ [rax + OFF\_SN], rsi

mov [rsi + OFF\_SP], rax

.endSuper

mov rax, [rdi + OFF\_R] ; Cambiar normales

mov rsi, [rdi + OFF\_N]

mov [rax + OFF\_N], rsi

mov [rsi + OFF\_P], rax

mov rax, [rdi + OFF\_D] ; Guardar \*info

push rax ; stack aligned

~~call~~ call Free ; Liberar nodo

pop rax

ret

b) agregar Super: ; rdin: euro, asumo modo de listosuper

```
mov pax, pdi ; pax ≤ cpr
```

~~prev~~, ~~p~~, ~~par~~;  $ps[i] \neq \text{nextS}(par)$

Find SP

```
mov    rax, [rax + OFF_P]
```

```
cmp [rax+OFF_SP], 0
```

12. find SP

*[Handwritten signature]*

~~MOV PS, [PAX + OFF, SM]~~

```
mov [Polx + OFF_SN], Pol
```

```
mov [rsi+OFF_SP], pdi
```

```
mov [Pd.off_sn], Psc
```

```
mov [rdi + OFF_SP], rdx
```

104

c) están Todos: ;  $rdi = xnode$

mov Psi, Pdi

Loop

CMP [PSI + OFF\_SN], 0

JNZ .is Super

XOP Pox, Pox

pet

- is Super

```
mov     PSI, [PSI+OFF_N]
```

Composi, Poli

JNE .loop

Nov Pöx, 1

pet

2/4

2. a) sumatorias ; ~~int\*\*~~sumatorias(~~int\*\*~~ m) ; Pdi = m

push Pdi ; stack aligned

mov Pdi, 64 ; 4 \* 4 \* 4b

call malloc ✓

pop Pdi ; Pdi = m, Pdx = ret

; Cargo todo

; xmm0-3: Fila ~~1~~ ~~xmm1-4: Fila 2~~ ~~xmm5-8: Fila 3~~ ~~xmm9-12: Fila 4~~

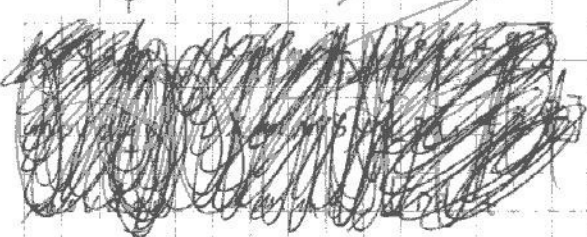


movdqu xmm0, [Pdi]

movdqu xmm1, [Pdi+16] ✓

movdqu xmm2, [Pdi+32]

movdqu xmm3, [Pdi+48]



; xmm4-7 = xmm8-11 = xmm0-3

movdqa xmm4, xmm0

movdqa xmm5, xmm1

movdqa xmm6, xmm2

movdqa xmm7, xmm3 ✓

movdqa xmm8, xmm0

movdqa xmm9, xmm1

movdqa xmm10, xmm2

movdqa xmm11, xmm3



; xmm4 = | ~~$x_{i,3}$~~ | $x_{i,2}$ | $x_{i,1}$ | $x_{i,0}$ | (can color)

paddb xmm4, xmm5

paddb xmm6, xmm7 ✓

paddb xmm4, xmm6

; xmm0 = | $x_{3,i}$ | $x_{2,i}$ | $x_{1,i}$ | $x_{0,i}$ |

phaddb xmm0, xmm1

phaddb xmm2, xmm3 ✓

phaddb xmm0, xmm2

; xmmi = | $x_{i,3}$ | $x_{i,2}$ | $x_{i,1}$ | $x_{i,0}$ |  $\forall 0 \leq i < 4$

psrldq xmm3, xmm0, 0b11111111

psrldq xmm2, xmm0, 0b10101010 ✓

psrldq xmm1, xmm0, 0b01010101

psrldq xmm0, xmm0, 0b00000000

; xmmi = |res<sub>i,3</sub> +  $x_{i,3}$ |res<sub>i,2</sub> +  $x_{i,2}$ |res<sub>i,1</sub> +  $x_{i,1}$ |res<sub>i,0</sub> +  $x_{i,0}$ |  $\forall 0 \leq i < 4$

paddb xmm0, xmm4

paddb xmm1, xmm4 ✓

paddb xmm2, xmm4

paddb xmm3, xmm4

; xmmi = |res<sub>i,3</sub>|res<sub>i,2</sub>|res<sub>i,1</sub>|res<sub>i,0</sub>|  $\forall 0 \leq i < 4$

psubd xmm0, xmm8

psubd xmm1, xmm9 ✓

psubd xmm2, xmm10

psubd xmm3, xmm11

add movdqu ~~[rax]~~, xmm0

movdqu [rax+16], xmm1 ✓

movdqu [rax+32], xmm2

movdqu [rax+48], xmm3

ret

3/4

Estoy procesando toda la matriz simultaneamente, no quedan mas elementos.

b) Para perder la menor precision posible debemos primero reemplazar las siguientes instrucciones

paddd  $\Rightarrow$  addps

phaddq  $\Rightarrow$  haddps

psubd  $\Rightarrow$  subps

y luego, en la etiqueta .dcd, agregar el siguiente código para convertir a enteros

cvttps2dq xmm0, xmm0

cvttps2dq xmm7, xmm7

cvttps2dq xmm2, xmm2

cvttps2dq xmm3, xmm3

Perderías menos precisión  
utilizando una conversión  
sin truncamiento, como  
cvtpps2dq.

3. 2) La nueva convención define dos pilas: DATA y ADDR.

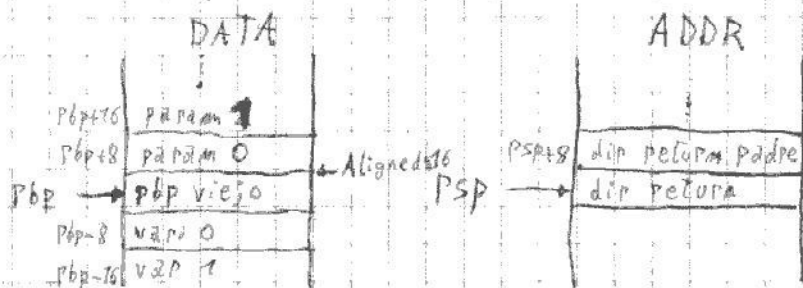
Siguiendo la convención C, que requiere ~~que~~ que el stack esté alineado a 16B para poder hacer lecturas alineadas de datos de ese tamaño, DATA debe estar alineado a 16B al llamar a una función. ADDR, al almacenar solo punteros, debe estar siempre alineado a 8B.

No se requieran variables globales para evitar incurrir en accesos extra a memoria. PSP siempre apunta a la dirección de retorno de la función actual, en ADDR.

El registro Pbp siempre apunta al valor de Pbp de la función padre, <sup>al ingresar a la función</sup> en DATA. Em  $[Pbp + 8 * (i+1)]$  se encuentran <sup>los</sup> ~~los~~ <sup>entre</sup> ~~entre~~ parámetros de la función, para  $i \geq 0$ . ~~La función puede guardar datos debajo de [Pbp].~~ El registro Pbp es caller-saved. DATA crece para abajo en memoria.

El orden de los parámetros fuera del stack es el mismo que en la convención C. Los registros caller/callee son igual que en la convención C, excepto por Pbp. No se debe usar PUSH/POP para el manejo de datos.

Ilustración al ~~al ingresar~~ <sup>al ingresar</sup> / antes de retornar:



~~Al~~ Al llamar a una función se debe agregar primero los parámetros a DATA. Luego se apila el Pbp actual (~~el Pbp actual~~) (debe quedar alineado) y se guarda su dirección en Pbp. Luego se ejecuta el CALL.

La función llamada ya tiene el stack-frame armado y termina simplemente



con un RET.

Al volver la función padre debe cargar [rbp] en rbp antes de retornar.

b) convert And Add; edi = 2, esi = 6

mov [rbp-8], esi

mov [rbp-16], ~~rbp~~

Sub? → add rbp, 16

call convert

esi? → mov edi, [rbp+8]

mov [rbp+8], eax

call convert

mov rbp, [rbp] ; o ADD rbp, 16

add eax, [rbp-8]

ret